WHAT IS CLAIMED IS:

1. A concurrent shared object representation comprising:
a computer readable encoding for a sequence of zero or more values; and
access operations defined for access to each of opposing ends of the sequence,
wherein execution of any one of the access operations is non-blocking with respect to any other execution of the access operations throughout a complete range of valid states, including one or more boundary condition states, and
wherein, at least for those of the valid states other than the one or more boundary condition states, opposing-end ones of the access operations are disjoint.

2. The concurrent shared object representation of claim 1,
wherein the computer readable encoding includes an array of elements for representing the sequence; and
wherein the one or more boundary condition states include a full state and an empty state.

3. The concurrent shared object representation of claim 1,
wherein the computer readable encoding includes a linked-list of nodes representing the sequence; and
wherein the one or more boundary condition states include one or more empty states.

4. The concurrent shared object representation of claim 1, wherein the access operations include push and pop operations.

5. The concurrent shared object representation of claim 4, wherein the access operations further include delete operations.

6. The concurrent shared object representation of claim 1, wherein the access operations include push and pop operations, including opposing end variants of each.

7. The concurrent shared object representation of claim 1, wherein the access operations include push and pop operations, including opposing end variants of at least one of the push and pop operations.

8. The concurrent shared object representation of claim 2,
wherein the array of elements is organized as a circular buffer of fixed size with opposing-end indices respectively identifying opposing ends of the sequence; and
wherein concurrent non-blocking access is mediated, at least in part, by performing, during execution of each of the access operations, an atomic update of a respective one of the opposing-end indices and of an array element corresponding thereto.

9. The concurrent shared object representation of claim 3,
wherein the access operations include push, pop and delete operations, and
wherein concurrent access is mediated, at least in part, by performing, during execution of each of the pop operations, an atomic update of a list node and both a deleted node indication and list-end identifier corresponding thereto.

10. The concurrent shared object representation of claim 9,
wherein concurrent access is further mediated, at least in part, by performing, during execution of each of the delete operations, an atomic update of a deleted node indication and at least one list-end identifier corresponding thereto.

11. The concurrent shared object representation of claim 3, wherein the linked-list of nodes is a doubly-linked list thereof.

12. A method of managing access to a dynamically allocated list susceptible to concurrent operations on a sequence encoded therein, the method comprising:
executing as part of a pop operation, an atomic update of a list node and both a deleted node indication and list-end identifier corresponding thereto;

the deleted node indication marking the corresponding element for subsequent deletion from the list.

13. The method of claim 12, further comprising:
executing as part of a delete operation, an atomic update of a deleted node indication and at least one list-end identifier corresponding thereto.

14. The method of claim 12, further comprising:
responsive to the deleted node indication, excising a marked node from the list by atomically updating opposing direction pointers impinging thereon and the deleted node indication thereto.

15. The method of claim 12, further comprising:
deleting the marked element from the list at least before completion of a same-end push or pop operation.

16. The method of claim 13,
wherein the list is a doubly-linked list susceptible to concurrent operation of opposing-end variants of the pop operation; and
wherein the atomic update includes execution of a DCAS.

17. The method of claim 13,
wherein the list is a doubly-linked list susceptible to concurrent operation of a same-end push operation; and
wherein the atomic update includes execution of a DCAS.

18. The method of claim 12, further comprising:
wherein the deleted node indication is encoded integral with an end-node identifying pointer.

19. The method of claim 12, further comprising:
wherein the deleted node indication is encoded as a dummy node.

20. A computer program product encoded in at least one computer readable medium, the computer program product comprising:

at least one functional sequence providing non-blocking access to on a concurrent shared object, the concurrent shared object instantiable as a linked-list delimited by a pair of end identifiers;

wherein instances of the at least one functional sequence concurrently executable by plural processors of a multiprocessor and each include an atomic operation to atomically update one of the end identifiers and a node of the linked-list corresponding thereto,

wherein for opposing end instances, the atomic updates are disjoint for at least all non-empty states of the concurrent shared object.

21. A computer program product as recited in 20, wherein the at least one functional sequence includes both push and pop functional sequences.

22. A computer program product as recited in 20,

wherein the at least one computer readable medium is selected from the set of a disk, tape or other magnetic, optical, or electronic storage medium and a network, wireline, wireless or other communications medium.

23. An apparatus comprising:

plural processors;

a store addressable by each of the plural processors;

first- and second-end identifier stores accessible to each of the plural processors for identifying opposing ends of a concurrent shared object in the addressable store; and

means for coordinating competing pop operations, the coordinating means employing in each instance thereof, an atomic operation to disambiguate a retry state and a boundary condition state of the concurrent shared object based on then-current contents of one, but not both, of the first- and second-end identifier stores and an element of the concurrent shared object corresponding thereto.